PROGRAMMING FOR PROBLEM SOLVING USING C
UNIT-V

Text Input / Output: Files, Streams, Standard Library Input / Output Functions, Formatting Input / Output Functions, Character Input / Output Functions, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Binary Input / Output: Text versus Binary Streams, Standard Library, Functions for Files, Converting File Type, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Functions: Designing, Structured Programs, Function in C, User Defined Functions, Inter-Function Communication, Standard Functions, Passing Array to Functions, Passing Pointers to Functions, Recursion, Passing an Array to Function, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

➢ Text Input / Output:

- **Files: -**
  - A file represents a sequence of bytes on the disk where a group of related data is stored.
  - File is created for permanent storage of data.
  - It is a ready-made structure.
  - You can use the fopen( ) function to create a new file or to open an existing file.
  - To close a file, use the fclose( ) function.
  - The function fputc() writes the character value of the argument c to the output stream referenced by fp.
  - The fgetc() function reads a character from the input file referenced by fp.

- **Streams:-**
  - In C all input and output is done with streams.
  - Stream is nothing but the sequence of bytes of data.
  - A sequence of bytes flowing into program is called input stream.
  - A sequence of bytes flowing out of the program is called output stream.
  - Use of Stream make I/O machine independent.

- **Standard Library I/O Functions: -**

- The standard input and output library is stdio.h, and you will find that you include this library in almost every program you write.
- It allows printing to the screen and collecting input from the user. The functions you will use the most include:
  - printf() is output to the screen
  - scanf() is read input from the screen
  - getchar() is return characters typed on screen
  - putchar() is output a single character to the screen
  - fopen() is open a file, and
  - fclose() is close a file

- **Formatting Input /Output Functions:-**
  - **Formatted Input**

    The function scanf() is used for formatted input from standard input and provides many of the conversion facilities of the function printf().

    Syntax

    scanf (format, num1, num2,……);
    The function scanf() reads and converts characters from the standards input depending on the format specification string and stores the input in memory locations represented by the other arguments (num1, num2,….).

    For Example:

    scanf(“ %c %d”,&Name, &Roll No);

- **Character Input / Output Functions:-**

Character input functions is used to input a single character.

All these functions are available in 'stdio.h' header file.

getch(): Use to input single character at a time. But it will not display input character. get stands for input, ch stands for character.
Syntax: char a = getch();

Character Output Functions**:-**
putch(): use to print a single character.
Syntax: putch(a);

putchar(): use to print a single character.
Syntax: putchar(a);

➢ Binary Input / Output:-

Text versus Binary Streams:-

Text:-

A text stream consists of one or more lines of text that can be written to a text-oriented display so that they can be read.

When reading from a text stream, the program reads an *NL* (newline) at the end of each line.

Writing to a text stream, the program writes an *NL* to signal the end of a line.

To match differing conventions among target environments for representing text in files, the library functions can alter the number and representations of characters transmitted between the program and a text stream.

Binary Stream:-

A binary stream consists of one or more bytes of arbitrary information.

You can write the value stored in an arbitrary object to a (byte-oriented) binary stream and read exactly what was stored in the object when you wrote it.

The library functions do not alter the bytes you transmit between the program and a binary stream.

They can, however, append an arbitrary number of null bytes to the file that you write with a binary stream.

The program must deal with these additional null bytes at the end of any binary stream.

- **Standard Library**:-

C Standard library functions or simply C Library functions are inbuilt functions in C programming.

The prototype and data definitions of these functions are present in their respective header files. To use these functions we need to include the header file in our program. For example,

If you want to use the printf() function, the header file <stdio.h> should be included.

```
#include<stdio.h>

int main()

{

printf("Catch me if you can.");

}
```

If you try to use printf() without including the stdio.h header file, you will get an error.

Advantages of Using C library functions

1. They work

One of the most important reasons you should use library functions is simply because they work. These functions have gone through multiple rigorous testing and are easy to use.

2. The functions are optimized for performance

Since, the functions are "standard library" functions, a dedicated group of developers constantly make them better. In the process, they are able to create the most efficient code optimized for maximum performance.

3. It saves considerable development time

Since the general functions like printing to a screen, calculating the square root, and many more are already written. You shouldn't worry about creating them once again.

4. The functions are portable

With ever-changing real-world needs, your application is expected to work every time, everywhere. And, these library functions help you in that they do the same thing on every computer.

Example: Square root using sqrt() function

Suppose, you want to find the square root of a number.

To can compute the square root of a number, you can use the sqrt() library function. The function is defined in the math.h header file.

```
#include<stdio.h>
#include<math.h>
int main()
{
floatnum, root;
printf("Enter a number: ");
scanf("%f", &num);

// Computes the square root of num and stores in root.

root = sqrt(num);

printf("Square root of %.2f = %.2f", num, root);
return0;
}
```

When you run the program, the output will be:

Enter a number: 12
Square root of 12.00 = 3.46

Library Functions in Different Header Files

C Header Files	
<assert.h>	Program assertion functions
<ctype.h>	Character type functions
<locale.h>	Localization functions
<math.h>	Mathematics functions
<setjmp.h>	Jump functions

C Header Files	
<signal.h>	Signal handling functions
<stdarg.h>	Variable arguments handling functions
<stdio.h>	Standard Input/Output functions
<stdlib.h>	Standard Utility functions
<string.h>	String handling functions
<time.h>	Date time functions

- **Functions for files**:-

C provides a number of functions that helps to perform basic file operations. Following are the functions,

Function	description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file

putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

- **Converting File Type:-**
  - Type conversion refers to changing a variable of one data type into another.
  - For instance, if you assign an integer value to a floating-point variable, the compiler will convert the int to a float.
  - Casting allows you to make this type conversion explicit, or to force it when it wouldn't normally happen.

➢ **Functions:-**

- **Designing:-**

Functions are an essential ingredient of all programs, large and small, and serve as our primary medium to express computational processes in a programming language. So far, we have discussed the formal properties of functions and how they are applied. We now turn to the topic of what makes a good function. Fundamentally, the qualities of good functions all reinforce the idea that functions are abstractions.

- Each function should have exactly one job. That job should be identifiable with a short name and characterizable in a single line of text. Functions that perform multiple jobs in sequence should be divided into multiple functions.
- *Don't repeat yourself* is a central tenet of software engineering. The so-called DRY principle states that multiple fragments of code should not describe redundant logic. Instead, that logic should be implemented once, given a name, and applied multiple times. If you find yourself copying and pasting a block of code, you have probably found an opportunity for functional abstraction.

- Functions should be defined generally. Squaring is not in the Python Library precisely because it is a special case of the pow function, which raises numbers to arbitrary powers.

These guidelines improve the readability of code, reduce the number of errors, and often minimize the total amount of code written. Decomposing a complex task into concise functions is a skill that takes experience to master. Fortunately, Python provides several features to support your efforts.

- **Structured programs:-**
  Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

- **Function in C:-**

A function is a block of statements that performs a specific task. Suppose you are building an application in C language and in one of your program, you need to perform a same task more than once. In such case you have two options –

a) Use the same set of statements every time you want to perform the task
b) Create a function to perform that task, and just call it every time you need to perform that task.

Using option (b) is a good practice and a good programmer always uses functions while writing codes in C.

Types of functions

1) **Predefined standard library functions** – such as puts(), gets(), printf(), scanf() etc – These are the functions which already have a definition in header files (.h files like stdio.h), so we just call them whenever there is a need to use them.

- 2) **User Defined functions –** The functions that we create in a program are known as user defined functions.

- **Inter-Function Communication:-**

When a function gets executed in the program, the execution control is transferred from calling a function to called function and executes function definition, and finally comes back to the calling function. In this process, both calling and called functions have to communicate with each other to exchange information. The process of exchanging information between calling and called functions is called inter-function communication.

In C, the inter function communication is classified as follows...

- **Downward Communication**
- **Upward Communication**
- **Bi-directional Communication**

- **Standard library functions:-**

The standard library functions are built-in functions in C programming.

These functions are defined in header files. For example,

The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the stdio.h header file.
Hence, to use the printf() function, we need to include the stdio.h header file using #include <stdio.h>.

The sqrt() function calculates the square root of a number. The function is defined in the math.h header file.

- **Passing Array to Functions**:-

Just like variables, array can also be passed to a function as an argument. In this guide, we will learn how to pass the array to a function using call by value and call by reference methods.

To understand this guide, you should have the knowledge of following C Programming topics:
C – Array

Function call by value in C
Function call by reference in C

Passing array to function using call by value method

As we already know in this type of function call, the actual parameter is copied to the formal parameters.

```
#include<stdio.h>
voiddisp(charch)
{
printf("%c ",ch);
}
int main()
{
chararr[]={'a','b','c','d','e','f','g','h','i','j'};
for(int x=0; x<10; x++)
{
/* I'm passing each element one by one using subscript*/
disp(arr[x]);
}

return0;
}
```

Output:

a b c d e f g h i j

Passing array to function using call by reference

When we pass the address of an array while calling a function then this is called function call by reference. When we pass an address as an argument, the function declaration should have a pointer as a parameter to receive the passed address.

```
#include<stdio.h>
voiddisp(int*num)
{
printf("%d ",*num);
}

int main()
{
intarr[]={1,2,3,4,5,6,7,8,9,0};
for(inti=0;i<10;i++)
{
/* Passing addresses of array elements*/
disp(&arr[i]);
}
```

```
return0;
}
Output:
1234567890
```

How to pass an entire array to a function as an argument?

In the above example, we have passed the address of each array element one by one using a for loop in C. However you can also pass an entire array to a function like this:

Note: The array name itself is the address of first element of that array. For example if array name is arr then you can say that arr is equivalent to the &arr[0].

```
#include<stdio.h>
voidmyfuncn(int*var1,int var2)
{
        /* The pointer var1 is pointing to the first element of
         * the array and the var2 is the size of the array. In the
         * loop we are incrementing pointer so that it points to
         * the next element of the array on each increment.
         *
         */
for(int x=0; x<var2; x++)
{
printf("Value of var_arr[%d] is: %d \n", x,*var1);
/*increment pointer for next element fetch*/
var1++;
}
}

int main()
{
intvar_arr[]={11,22,33,44,55,66,77};
myfuncn(var_arr,7);
return0;
}
Output:
Value of var_arr[0]is:11
Value of var_arr[1]is:22
Value of var_arr[2]is:33
Value of var_arr[3]is:44
Value of var_arr[4]is:55
Value of var_arr[5]is:66
```

Value of var_arr[6]is:77

- **Passing pointer to function:-**

In this example, we are passing a pointer to a function. When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value. So any change made by the function using the pointer is permanently made at the address of passed variable. This technique is known as call by reference in C.

```
#include<stdio.h>
voidsalaryhike(int *var, int b)
{
   *var = *var+b;
}
int main()
{
int salary=0, bonus=0;
printf("Enter the employee current salary:");
scanf("%d", &salary);
printf("Enter bonus:");
scanf("%d", &bonus);
salaryhike(&salary, bonus);
printf("Final salary: %d", salary);
return0;
}
```

Output:

```
Enter the employee current salary:10000
Enter bonus:2000
Final salary: 12000
```